Mustererkennung - Aufgabenblatt 05

André Hacker und Dimitri Schachmann

1. Ridge Regression

Für diese Aufgabe haben wir zunächst die Daten entsprechend der Aufgabenstellung normiert:

```
% Standardize data by subtracting the mean and deviding by the standard
% deviation
function r = standardize(samples)
  m = mean(samples);
  sd = std(samples);
  for i = 1:size(samples,1)
    samples(i,:) = (samples(i,:) - m) ./ sd;
  end
  r = samples;
end
```

Das Einlesen der Daten sieht dann entsprechend wie folgt aus.

```
tra = standardize(dlmread('prostate-tra.mat', ' '));
trates = standardize(dlmread('prostate-all.mat', ' '));
```

Unsere Funktion für die Berechnung der ridge regression sieht wie folgt aus:

```
% Computes the ridge regression parameters
function a = ridgeRegression(y, samples, lambda)
  a = [];
  for i = 1:size(lambda,2)
    t = inv(samples'*samples + (lambda(:,i)*eye(size(samples'*samples))));
    a = [a (t * samples'*y)];
  end
end
```

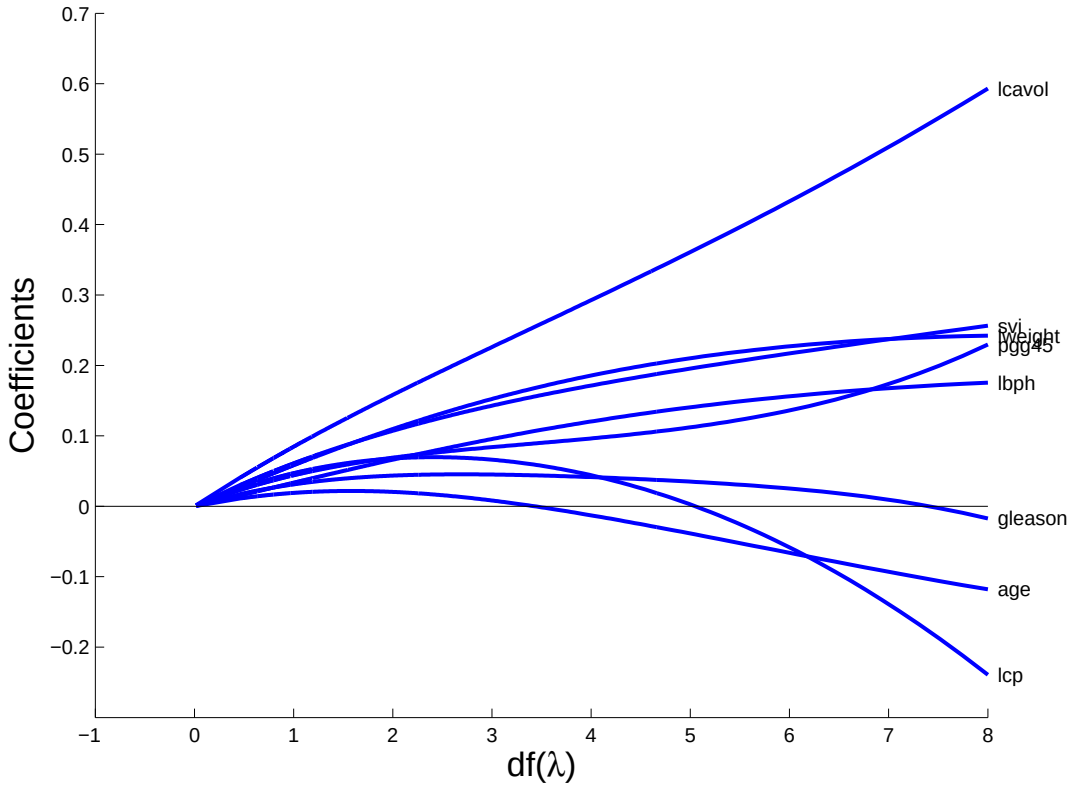
Schließlich haben wir das Ergebnis grafisch dargestellt mit der folgenden funktion:

```
% plot the ridge regression
function r = plotRidgeRegression(y, samples, labels, filename)
  k = [0:1:1000 30000];
  data = ridgeRegression(y,samples,k);
  plt = figure();
  xlim([-1 8]);
  ylim([-0.3 0.7])
  hold on;
  plot(df(samples,k), data,'LineWidth',2,'Color','blue');
  for i = 1:size(labels,1)
    text(size(data,1)+0.1, data(i,1) ,labels(i,:),'FontSize',10);
  end
  line([-1 8],[0 0], 'Color', 'black')
  xlabel('df(\lambda)', 'FontSize', 17);
  ylabel('Coefficients', 'FontSize', 17);
  print(plt,'-depsc',[filename '.eps']);
end
```

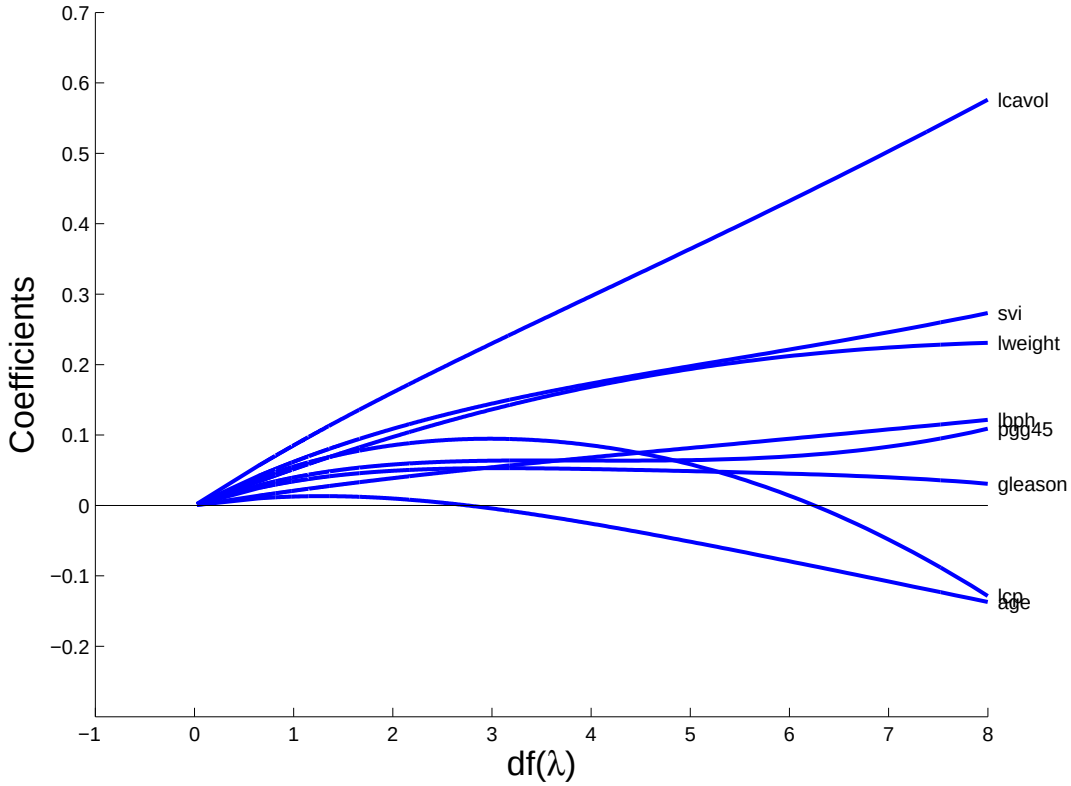
Die Funktion für die effektiven Freiheitsgrade haben wir von Hastie (2011):

```
% df(l) function from Hastie
function r = df(samples, lambda)
  r = [];
  for i = 1:size(lambda,2)
    t = inv(samples'*samples + (lambda(:,i)*eye(size(samples'*samples))));
    a = samples * t * samples';
    r = [r trace(a)];
  end
end
```

Das Ergebnis sieht dann mit den Trainingsdaten so wie im Buch aus:



Wenn wir alle Samples verwenden, dann sieht das wie folgt aus:



2. Bootstrap

Für den Bootstrap Algorithmus lesen wir alle prostate Daten aus und wählen dort nur die Merkmale 1, 5 und 7 aus.

```
% read the ALL of the data
all = dlmread('prostate-all.mat', ' ');

% we consider only features 1, 5 and 7
testn = all(1:10,[1 5 7]);

% Add 1 dimension for offset/bias
testn = [ones(size(testn, 1),1) testn];
```

Als nächstes wird in einer Schleife für verschiedene Proben der Bootstrap Algorithmus durchgeführt:

```
q = 50;   % number of sample picks per iteration
bootstraps = [];
for i = 1:100
  % pick 50 random indecies
  ran = randi([1,size(all,1)],q,1);
  % get the samples at the random indecies
  allIn  = all(ran,[1 5 7]);
  % get the actual values at the indecies
  allOut = all(ran,9);
  % Add 1 dimension for offset/bias
  allIn = [ones(size(allIn, 1),1) allIn];
  % compute linear regression and add to bootstrap list
```

```
    bootstraps = [bootstraps getWeightsLeastSquares(allIn, allOut)];
  end
```

Für die Berechnung der linearen regression verwenden wir unsere alte Funktion:

```
% Least Squares Fitting based on samples and output column vector
function w = getWeightsLeastSquares(samples, y)
  % Compute pseudo-inverse matrix
  pseudo = getPseudoInverse(samples);
  w = pseudo * y;
end

% Get pseudo-inverse matrix, needed for least squares fitting
function p = getPseudoInverse(X)
  % very likely that inverse exists if we have many samples
  p = inv(X' * X) * X';
end
```

Für die Bestimmug des Mittelwertes und des Konfidenzintervals für verschiedene Proben, haben wir eine Funktion geschrieben:

```
% use bootstraped data for prediction
function [m d] = computeConfidence(bootstraps, y)
  m = [];
  d = [];
  for i = 1:size(y,1)
    p = [];
    for j = 1:size(bootstraps,2)
      p = [p predict(bootstraps(:,j), y(i,:))];
    end
    m = [m ; mean(p)];
    d = [d ; 2*std(p)];
  end
end

% Predicts value for input based on weights
% samples = list of row vectors
% weights = column-vector
function r = predict(weights, samples)
  r = samples * weights;
end
```

Das haben wir dann auf die ersten 10 Samples aus den Daten angewendet.

```
  [m d] = computeConfidence(bootstraps, testn);
  fid = fopen('task2-results.txt','w');
  fprintf(fid, 'Mean and 2*standard deviation for the first %d samples\n', ...
          size(testn,1));
  for i = 1:size(testn,1)
    fprintf(fid, 'Sample %d\t| Mean = %f | confidence =  %f\n', i, m(i,:),d(i,:));
  end
```

Das Ergebnis davon ist im folgenden zu sehen:

```
Mean and 2*standard deviation for the first 10 samples
Sample 1 | Mean = 1.190903 | 2 x standard deviation =  0.426602
Sample 2 | Mean = 0.954483 | 2 x standard deviation =  0.494090
Sample 3 | Mean = 1.323101 | 2 x standard deviation =  0.483985
Sample 4 | Mean = 0.834844 | 2 x standard deviation =  0.531425
Sample 5 | Mean = 1.950328 | 2 x standard deviation =  0.322340
Sample 6 | Mean = 0.922782 | 2 x standard deviation =  0.503808
Sample 7 | Mean = 1.942197 | 2 x standard deviation =  0.322115
Sample 8 | Mean = 1.917087 | 2 x standard deviation =  0.321625
Sample 9 | Mean = 1.078686 | 2 x standard deviation =  0.457431
Sample 10 | Mean = 1.648966 | 2 x standard deviation =  0.335726
```

3. Experiment

Wir haben das Experiment wie beschrieben durchgeführt, also einen zufälligen 16×1 Vektor wiederholt mit der Kovarianzmatrix einer Ziffer multipliziert. Ziwschendurch haben wir jedes mal auf 1 normiert. Hier beispielsweise für die Ziffer 8:

```
eight = load('pendigits8.txt');
eight = eight(:,1:end-1);
[m covarianceMatrix] = meanAndCov(eight);
covarianceMatrix = covarianceMatrix/norm(covarianceMatrix);
ran = randi([1,100],16,1)';
ran = ran/norm(ran);
for i = 1:10
  ran = ran * covarianceMatrix;
  ran = ran/norm(ran);
end
```

In der Vorlesung haben wir gelernt, dass ran jetzt den längsten Eigenvektor der Kovarianzmatrix enthalten sollte. Das haben wir mit der eig() Funktion von Matlab überprüft:

```
[eigenvectors, eigenvalues] = eig(covarianceMatrix);
pc =eigenvectors(:,end)';
fid = fopen('task3-results.txt','w');
fprintf(fid, 'experiment. result \t| longest eigenvector\n');
fprintf(fid, '-----------------------------------------\n');
for i = 1:size(ran,2)
  fprintf(fid, '%+.3f\t\t|\t%+.3f\n',ran(:,i),pc(:,i));
end
```

Und tatsächlich sieht das Ergebnis wie folgt aus:

```
experiment. result  | longest eigenvector
-------------------------------------
-0.272 | -0.272
+0.011 | +0.011
+0.013 | +0.013
-0.229 | -0.229
+0.295 | +0.295
-0.286 | -0.286
-0.189 | -0.189
-0.190 | -0.190
-0.214 | -0.214
+0.171 | +0.171
+0.394 | +0.394
+0.283 | +0.283
+0.276 | +0.276
+0.260 | +0.260
-0.420 | -0.420
+0.036 | +0.036
```